01-669
1496.00169

HARDWARE SEMAPHORES FOR A MULTI-PROCESSOR SYSTEM WITHIN A SHARED MEMORY ARCHITECTURE

Field of the Invention

The present invention relates to a method and/or architecture for implementing semaphores generally and, more particularly, to a method and/or architecture for hardware semaphores within a multi-processor system.

Background of the Invention

A conventional processor executing real-time operating systems commonly executes several software programs in a time-sliced fashion. The software programs therefore compete with each other for use of system resources. Resource contentions are commonly prevented by employing semaphores between the software tasks. Software-based semaphores are created within the software by progressing through a section of "critical" code where task switching is prevented and memory (where a software semaphore would be stored) accessed while all task switching/interrupt servicing is suspended. Executing the "critical" code can involve servicing

exceptions that are relatively time consuming and processor intensive.

The resource contention problem grows significantly when multiple processors are used in the system implementing a unified memory mapping. Each processor commonly requires access to shared resources such as shared regions of memory for inter-processor command passing and shared use of system peripherals such as a Data Move Engine with Direct Memory Access capabilities. Since each task and processor will function independently, a system of semaphores is required to ensure that operations are sequenced correctly (i.e., while a resource is in use, an active semaphore ensures that no other task uses the resource).

Multi-processor software semaphore implementations commonly require some form of communication between processors to lock out all processors from the shared resource while a semaphore is written. The inter-processor communication in itself requires semaphores to control the shared memory necessary for command passing and so the system becomes unstable. Running the critical code to set the semaphores conventionally involves locking out all system interrupts and so can cause timing issues with regard to true real-time operation. A resource shared between processors

requires that the sharing processors be locked within a safe state. In other words, the sharing processors must not switch tasks or service interrupts.

Summary of the Invention

The present invention concerns a circuit generally comprising a memory element and a controller. The memory element may define a semaphore allocatable to a resource. The controller may be configured to (i) present a granted status in response to a processor reading a first address while the semaphore has a free status, (ii) set the semaphore to a busy status in response to presenting the granted status, and (iii) present the busy status in response to the processor reading the first address while the semaphore has the busy status.

The objects, features and advantages of the present invention include providing a method and/or architecture for hardware semaphores within a multi-processor system that may (i) operate independently of critical sections of code, (ii) provide read-modify-write access patterns to both request and set the semaphores within a single operation, (iii) provide a more robust solution than the software semaphore equivalent, (iv) minimize

processor involvement in setting and resetting the semaphores, (v) permit allocation and reallocate of the semaphores to the resources as needed, and/or (vi) cascade the semaphores so that a semaphore may be used to mark another set of semaphores as "in use".

Brief Description of the Drawings

These and other objects, features and advantages of the present invention will be apparent from the following detailed description and the appended claims and drawings in which:

FIG. 1 is a block diagram of a system incorporating the present invention;

FIG. 2 is a block diagram of a semaphore circuit implementing a preferred embodiment of the present invention;

FIG. 3 is a drawing of semaphore data as seen by a processor;

FIG. 4 is a state transition diagram;

FIG. 5 is a block diagram of a second embodiment of the semaphore circuit;

FIG. 6 is a second state transition diagram; and

FIG. 7 is a third state transition diagram.

Detailed Description of the Preferred Embodiments

Referring to FIG. 1, a block diagram of a system 100 is shown incorporating the present invention. The system 100 may be implemented as a multi-processor system within a shared memory architecture. The system 100 may include system-on-chip designs incorporating a unified memory map architecture, board level designs, multi-board box level designs, and the like.

The system 100 generally comprises one or more processors 102A-D, a memory circuit 104, one or more resources 106A-N, a circuit 108, and a bus 110. The processors 102A-D may communicate with each other, the memory circuit 104, the resources 106A-N and the circuit 108 on the bus 110. The system 100 may include an optional second bus 112. The processors 102A-D may communicate with each other, the memory circuit 104, and/or the circuit 108 on the second bus 112.

The processors 102A-D may be executing multiple software tasks (not shown) in a real-time operating system environment. The memory circuit 104 may be implemented as a shared memory circuit accessible to some or all of the processors 102A-D. The resources 106A-N may be implemented as a variety of shared hardware and/or software resources of the system 100. Examples of the resources

106A-N may include, but are not limited to, mass memory controllers, communication circuits, input/output circuits, video processors, audio processors, network interface circuits, memory circuits (e.g., the memory circuit 104), and the like. The bus 110 may be implemented as a main or system type bus. The bus 112 may be implemented as a local or private type bus.

The circuit 108 may be implemented as a dedicated hardware block having multiple circuits 114A-T. Each circuit 114A-T may be implemented as a semaphore circuit defining a semaphore. Each semaphore circuit 114A-T may appear to the rest of the system 100 as one or more system registers with read-modify-write capabilities. Each semaphore may be configured to be set and reset within a single memory access (e.g., a read or a write) to a respective semaphore circuit 114A-T.

The semaphores may be defined independently of the resources 106A-N. There may be fewer, as many as, or more semaphores as resources 106A-N available in the system 100. Allocation and re-allocation of the semaphores to the resources 106A-N may be provided by the software as required. A semaphore may also be cascaded to mark another set of one or more semaphores

as "in use". Therefore, the semaphores of the circuit 108 may also be considered as part of the resources 106A-N.

Referring to FIG. 2, a block diagram of an example semaphore circuit 114K implementing a preferred embodiment of the present invention is shown. All of the semaphore circuit 114A-T may have a common design to the semaphore circuit 114K. The semaphore circuit 114K may interface with either the system bus 110 or the second bus 112. For the purposes of discussion, the semaphore circuit 114K may be assumed to be interfaced with the system bus 110.

The semaphore circuit 114K generally comprises a controller circuit 116 and a memory element 118. The controller circuit 116 may present a signal (e.g., WSEM) to the memory element 118). The memory element 118 may present a signal (e.g., RSEM) to the controller circuit 116. The signal WSEM may be implemented as a semaphore status written to the memory element 118. The signal RSEM may be implemented as a semaphore status read from the memory element 118.

The controller circuit 116 may be configured to operate as a register 120 and a register 122 as seen from the system bus 110. The register 120 may be implemented as a status register

accessible at an address. The register 122 may be implemented as a debug register accessible at another address.

The controller circuit 116 may receive a signal (e.g., ADDR) from the system bus 110. The controller circuit 116 may receive another signal (e.g., R/W) from the system bus 110. The registers 120 and 122 may exchange a signal (e.g., DATA) with the system bus 110.

The signal ADDR may be implemented as an address signal. The signal R/W may be implemented as a read/write signal. The signal DATA may be implemented as a data signal. The signal DATA may convey the semaphore status and other information.

The controller circuit 116 may respond to the signals ADDR, R/W, DATA, and RSEM to present the signal WSEM with a free status (value) or a busy status (value). The status conveyed by the signal WSEM may then be written into the memory element 118 as the status of the semaphore. The controller circuit 116 may respond to the signals ADDR, R/W, and RESM to present the signal DATA with the free status, the busy status, or a granted status (value). The signal DATA may then convey the semaphore status to the processors 102A-D.

Referring to FIG. 3, a drawing of the semaphore within the signal DATA is shown. The signal DATA may be implemented to match a width of the system bus 110. For example, the signal DATA may be implemented as a 32-bit wide signal. The semaphore may be embedded within two bits of the signal DATA, in particular, a bit zero and a bit one (e.g., DATA[1:0]). The remaining thirty bits of the signal DATA (e.g., DATA[31:2]) may be set to a logical zero value to simplify software testing. Other implementations of the signal DATA may be provided to meet the design criteria of a particular application. Status of the semaphore may be indicated by the signal DATA through the status register 120 as shown in TABLE 1 below:

TABLE 1

Semaphore Status	Value Bit 1	Value Bit 0
Granted	0	1
Busy	0	0
Free	1	0
Invalid Combination	1	1

Status of the semaphore may be indicated by the signal DATA through the debug register 122 as shown in TABLE 2 below:

TABLE 2

Semaphore Status	Value Bit 1	Value Bit 0
Invalid Combination	0	1
Busy	0	0
Free	1	0
Invalid Combination	1	1

Referring to FIG. 4, a state transition diagram 124 of the semaphore status is shown. Transitions in the state transition diagram 124 may be expressed as "trigger event/response". A phrase to the left of the slash "/" may define the trigger event that causes the transition to take place. A phrase to the right of the slash "/" may define the response, if any, presented due to the transition.

The semaphore may be determined by the logical state of the memory element 118. The semaphore may have either the free status (e.g., FREE state) or the busy status (e.g., BUSY1 state).

Access to the semaphore by the processors 102A-D may be performed via the status register 120 and the debug register 122.

A discussion of the state transition diagram 124 may begin assuming that the semaphore has the free status and thus the state transition diagram 124 is in the FREE state. A write of any value by a processor 102A-D to the address of the status register 120 may trigger a transition 126. The transition 126 may maintain the FREE state and may not present any response. A read by a processor 102A-D to the address of the status register 120 may trigger a transition 128 to the BUSY1 state. A response to the transition 128 may be to present the signal DATA through the status register 120 with the granted status. Another response to the transition 126 may be to set the memory element 118 to the busy status with the signal WSEM.

From the BUSY1 state, any additional reads to the status register 120 may produce a transition 130. The transition 130 may maintain the BUSY1 state and present the busy status through the signal DATA. A write of any value to the status register 120 while in the BUSY1 state may trigger a transition 132. The transition 132 may transition the semaphore circuit 114K back to the FREE

state. The transition 132 may also set the memory element 118 to the free status through the signal WSEM.

The processors 102A-D and any other master on the system bus 110 may read from and write to the debug register 122 without changing the semaphore status. Therefore, the debug register 122 may allow for non-intrusive examinations of the semaphores. From the FREE state, a read to the debug register 122 may trigger a transition 134. The transition 134 may maintain the FREE state and present the free status in the signal DATA. A write of any value to the debug register 122 may trigger a transition 136. The transition 136 may maintain the FREE state and not present any response.

From the BUSY1 state, a read of the debug register 122 may trigger a transition 138. The transition 138 may maintain the BUSY1 state and present the busy status through the signal DATA. A write of any value to the debug register 122 may trigger a transition 140. The transition 140 may maintain the BUSY1 state and not present any response.

As mentioned earlier, a semaphore may be considered one of the resources 106A-N that may be allocated to another semaphore. When semaphores are cascading, several different relationships may

be provided among a "master" semaphore and the other "slave" semaphores being marked. For example, changing the status of the master semaphore may have no impact to the status or memory elements 118 of the slave semaphores. As a result, each processor 102A-D may need to understand that a non-interactive relationship exists between the semaphores. A read of the slave semaphore marked as busy by the master semaphore may still return the free status. Likewise, a read of the slave semaphore marked as free by the master semaphore may still return the busy status. A second master/slave relationship may force the busy status for the slave semaphore whenever the master semaphore transitions from the free status to the busy status. When the master semaphore transitions from the busy status to the free status, the second master/slave relationship may also transition the slave semaphore to the free status. A third relationship may return the slave semaphore to a prior status just before the master semaphore transitioned to the busy status.

Referring to FIG. 5, a block diagram of a second embodiment of a semaphore circuit 114K' is shown. The semaphore circuit 114K' may be configured to permit slave semaphores to actively respond to changes in the master semaphores. The

semaphore circuit 114K' generally comprises a controller circuit 116', the memory element 118, and a multiplexer 142. The controller circuit 116' may receive the signals ADDR, R/W, and RSEM and present the signals DATA and WSEM. In addition, the semaphore circuit 114K' may have a plurality of lines 144A-T that interconnect some or all of the semaphore circuits 114A'-T'.

The multiplexer 142 may have inputs to receive each signal RSEM on lines 144A'-T' except for the local signal RSEM (e.g., from the semaphore circuit 114K'). The input to the multiplexer 142 for the local signal RSEM may be hardwired to the free status. The multiplexer 142 may present a multiplexed signal (e.g., MST) to the controller circuit 116'. The signal MST may indicate a master semaphore to which the local semaphore (e.g., K) has been cascaded or slaved.

The controller circuit 116' may be further configured to have another register 146 visible on the system bus 110. The register 146 may be implemented as a master register. The master register 146 may present a signal (e.g., SEL) to the multiplexer 142. The signal SEL may be implemented as a select signal used by the multiplexer 142 to select which signal RSEM is presented as the signal MST.

The processors 102A-D may use the signals ADDR, R/W and DATA to control a master/slave relationship for the semaphore circuit 114K'. Writing a value for a semaphore circuit 114B' into the master register 146 of the semaphore circuit 114K' generally establishes the semaphore circuit 114B' as the master and the semaphore circuit 114K' as the slave. Writing a value for the semaphore circuit 114K' into the master register 146 of the semaphore circuit 114K' generally creates no master/slave relationship (e.g., the semaphore circuit 114K' is master to itself).

Referring to FIG. 6, a state transition diagram 148 for a controller circuit 116' of a slave semaphore circuit 114S' is shown. The state transition diagram 148 shows an example where the semaphore circuit 114S' has been actively slaved to a master semaphore circuit 114M'. The left half of the state transition diagram 148 may be identical to and operate the same as the state transition diagram 124 shown in FIG. 4. The state transition diagram 148 may include an additional busy state (e.g., BUSY2).

A read to the status register 120 of the master semaphore circuit 114M' may trigger a transition 150 from the FREE state to the BUSY2 state for the slave semaphore circuit 114S'. A response

to the transition 150 may be to set the memory element 118 in the slave semaphore circuit 114S' to the busy state. Any subsequent read of the status register 120 of the slave semaphore circuit 114S' may trigger a transition 152. The transition 152 may leave the slave semaphore circuit 114S' in the BUSY2 state and respond by presenting the busy status.

A write to the status register 120 of the slave semaphore circuit 114S' while in the BUSY2 state may trigger a transition 154. The transition 154 may maintain the BUSY2 state without presenting a response. A read of the debug register 122 of the slave semaphore circuit 114S' may trigger a transition 156. The transition 156 may maintain the BUSY2 state and respond by presenting the busy status. A write to the debug register 122 of the slave semaphore circuit 114S' may cause the transition 158. The transition 158 may maintain the BUSY2 state without producing a response.

A write to the status register 120 of the master semaphore circuit 114M' may cause the master semaphore to transition to the free status and trigger a transition 160 for the slave semaphore. The transition 160 may change the slave semaphore circuit 114M' from the BUSY2 state to the FREE state. A response

to the transition 160 may also to set the memory element 118 in the slave semaphore circuit 114S' to the free status via the signal WSEM.

A transition of the master semaphore circuit 114M' to the busy status while the slave semaphore circuit 114S' is in the BUSY1 state may trigger a transition 162. The transition 162 may change to the slave semaphore circuit 114S' from the BUSY1 state to the BUSY2 state. The transition 162 may not produce a response since the memory element 118 in the slave semaphore circuit 114S' is already storing the busy status.

An effect of BUSY2 state may be to force the slave semaphore to have the busy status whenever the master semaphore transitions to the busy status. Another effect of the BUSY2 state may be to transition the slave semaphore to the free status whenever the master semaphore transitions from the busy status to the free status. In one embodiment, the slave resource may be configured to always have the same status as the master resource by allocating the slave resource directly to the same semaphore as the master resource.

Referring to FIG. 7, another state transition diagram 164 for the controller circuit 116' of the slave semaphore circuit

114S' is shown. The state transition diagram 164 is similar to the state transition diagram 148 except for the transition 162. The transition 162 may transition the slave semaphore circuit 114S' from the BUSY1 state to another busy state (e.g., BUSY3). The BUSY3 state may allow the slave semaphore circuit 114S' to retain a history of being in the BUSY1 state when the master semaphore transitioned from the free status to the busy status.

From the BUSY3 state, a write to the status register 120 of the master semaphore circuit 114M' may cause a transition 165 in the slave semaphore circuit 114S'. The transition 165 may transition the slave semaphore circuit 114S' from the BUSY3 state to the BUSY2 state. No response may be necessary for the transition 165 since the memory element 118 of the slave semaphore circuit 114S' already stores the busy status.

A read of the status register 120 of the slave semaphore circuit 114S' while in the BUSY3 state may trigger a transition 166 that maintains the BUSY3 state and presents the busy status as the response. A write to the statue register 120 of the slave semaphore circuit 114S' may trigger a transition 168 that maintains the BUSY3 state without a response. A read of the debug register 122 of the slave semaphore circuit 114S' may trigger a transition

170 that maintains the BUSY3 state and presents the busy status as the response. A write to the debug register 122 of the slave semaphore circuit 114S' state may trigger a transition 172 that maintains the BUSY3 state with no response.

In one embodiment, the transitions 152 and 168 may be altered to permit transitions between the BUSY2 state and the BUSY3 state. The read from the status register 120 of the slave semaphore circuit 114S' while in the BUSY2 state may trigger a transition 152'. The transition 152' may change the slave semaphore circuit 114S' from the BUSY2 state to the BUSY3 state and produce the busy status as the response. The write to the status register 120 of the slave semaphore circuit 114S' while in the BUSY3 state may trigger a transition 168'. The transition 168' may change the slave semaphore circuit 114S' from the BUSY3 state to the BUSY2 state without producing a response.

Other master/slave relationships may be implemented to meet the design criteria of a particular application. For example, the semaphore circuits 114A-T may be modified so that the master semaphores are aware of and responsive to the status of the slave semaphores. The master semaphore may thus be in the FREE state only when all of the slaved semaphores are in the FREE state. In

another example, the invalid "11" bit combination of the semaphore may be defined to indicate that the semaphore is slaved to another semaphore. Other variations of the semaphore encoding within the signal DATA may be implemented to meet the design criteria of a particular application.

The various signals of the present invention are generally "on" (e.g., a digital HIGH, or 1) or "off" (e.g., a digital LOW, or 0). However, the particular polarities of the on (e.g., asserted) and off (e.g., de-asserted) states of the signals may be adjusted (e.g., reversed) accordingly to meet the design criteria of a particular implementation. Additionally, inverters may be added to change a particular polarity of the signals.

While the invention has been particularly shown and described with reference to the preferred embodiments thereof, it will be understood by those skilled in the art that various changes in form and details may be made without departing from the spirit and scope of the invention.